Distributed Enforcement of Service Choreographies

Marco Autili Massimo Tivoli
Department of Information Engineering Computer Science and Mathematics
University of L'Aquila - ITALY
marco.autili@univaq.it massimo.tivoli@univaq.it

Modern service-oriented systems are often built by reusing, and composing together, existing services distributed over the Internet. Service choreography is a possible form of service composition whose goal is to specify the interactions among participant services from a global perspective. In this paper, we formalize a method for the distributed and automated enforcement of service choreographies, and prove its correctness with respect to the realization of the specified choreography. The formalized method is implemented as part of a model-based tool chain released to support the development of choreography-based systems within the EU CHOReOS project. We illustrate our method at work on a distributed social proximity network scenario.

1 Introduction

The future trend in service-oriented development is to build systems by reusing, and composing together, existing services distributed over the Internet.

Service choreography is a form of service composition whose goal is to specify message exchanges among multiple partner services, called *participants*, from a global perspective. When building a service-based system, a possible engineering approach is to compose services in a distributed way by considering this global specification. To this extent, the following two problems are usually considered: (a) *realizability check* – checks whether the choreography can be realized by implementing each participant so that it conforms to the choreography role that specifies its "expected" behaviour; and (b) *conformance check* – checks whether the global interaction of a set of services satisfies the choreography. In the literature many approaches have been proposed to address these problems, e.g., [11, 37, 35, 7, 8]. However, when the goal is to actually realize a service choreography by reusing *third-party* services, hence going beyond just checking its effectiveness, a further problem worth to be considered concerns automated choreography *enforcement*. That is, how to coordinate the interactions among participant services in order to fit the choreography specification. This requires to distribute and enforce, among the participants, the global coordination logic extracted from the choreography specification. The general problem here is that, although services may have been discovered or registered as suitable participants, their composite interaction may prevent the choreography realization if left uncontrolled (or wrongly coordinated).

BPMN2[1] *Choreography Diagrams* represent a de facto standard in the current practice of choreography specification and design. Within the CHOReOS EU project[2], the work presented in this paper is part of, we implemented a model-based transformation[3] to synthesize, out of a BPMN2 choreography diagram, an intermediate state-based model called *Choreography explicit-Flow Model* (CeFM). The latter is a choreography model that, conforming to the BPMN2 standard specification, makes explicit

[1] http://www.omg.org/spec/BPMN/2.0
[2] http://www.choreos.eu
[3] See documentation available at choreos.disim.univaq.it for details.

Javier Cámara and José Proença (Eds.):
13th International Workshop on Foundations of
Coordination Languages and Self-Adaptive Systems (FOCLASA'14)
EPTCS 175, 2015, pp. 18–35, doi:10.4204/EPTCS.175.2

coordination-related information that in BPMN2 is implicit. This allows to statically infer the information needed for enabling distributed coordination that, otherwise, should be calculated at run time for each choreography instance and for each execution of it. For instance, the CeFM model specifies the source and target state from which a task is initiated and terminated, the corresponding transition and enabling condition.

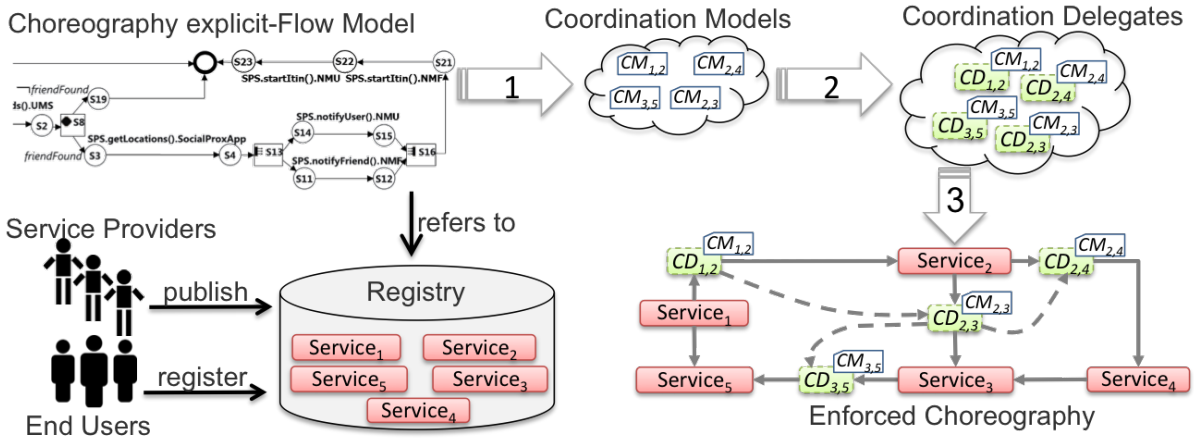

Figure 1: Overview of the choreography enforcement method

Contribution of the paper – In this paper, we formalize a choreography enforcement method based on distributed coordination of the participants from outside. Figure 1 shows an overview of our method as organized into three steps. Out of a CeFM *C*, our method automatically derives a set of what we call *Coordination Delegates* (CDs). In particular, the coordination logic modeled by *C* is distributed into a set of *Coordination Models* (CMs), one for each CD (step 1 in the figure). The CDs exploit the CMs in order to coordinate the interaction of the participants in a way that the resulting collaboration realizes the choreography specified by *C* (step 2 in the figure). The derived CDs are interposed (only when strictly needed) among the participants according to the CHOReOS architectural style [3, 14] (step 3 in the figure). To give an intuition, Figure 1 uses a box/arrow (i.e., component/connector) notation to show a sample architecture instance conforming to the CHOReOS style (see the bottom-most diagram in the right-hand side of the figure). Within CHOReOS, our method refers to a service registry to discover services that are suitable to play the roles of the choreography. The registry contains services published by providers that have identified business opportunities in the domain of interest. The registry also contains the registration of the end users interested in exploiting the choreography through client applications.

CDs perform pure business-level coordination by proxifying the service interaction, and mediate it by exchanging the coordination information contained in their own CMs, in a fully distributed way. In this way, CDs prevent possible *undesired interactions*. The latter are those interactions that do not belong to the set of interactions allowed by *C* and can happen when the services collaborate in an uncontrolled way. We formalize the notions of CeFM and CM, and show how to decompose a CeFM into a set of CMs. Furthermore, we describe the distributed coordination algorithm that is implemented by the CDs to realize the coordination logic.

The overall enforcement approach has been implemented as a set of REST services, and a graphical Eclipse plugin that invokes them has been released[4]. A tool demo screencast is also available.

[4] `choreos.disim.univaq.it` and `www.ow2.org/view/Future_Internet`

Advance with respect to our previous work – The work described in this paper represents an advance with respect to our previous work in [3]. In fact, although the synthesis process described there treats most of the BPMN2 constructs, it considers a simplified version of their actual semantics. For instance, the selection of conditional branches is simply abstracted as a non-deterministic choice, regardless the run-time evaluation of their enabling conditions. Analogously, parallel flows are enforced by non-deterministically choosing one of their linearizations obtained through interleaving, hence loosing the actual parallelism degree. To overcome these limitations, in this paper, we formalize the CeFM model that, being more expressive than the choreography model adopted in [3], preserves the actual semantics of the BPMN2 constructs. Relying on the CeFM model has led us to define a novel and more effective distributed coordination algorithm. As a further advance with respect to [3], we also prove correctness of the algorithm with respect to preventing undesired interactions. In particular, the proof gives a rigorous characterization of the notion of undesired interaction that in [3] has been informally treated hence making it difficult to effectively assess our method.

Main focus of the paper – It is worth to mention that the coordination logic performed by the CDs is *service-independent* since it is based on the expected behaviour of the participants as specified by C, rather than on the actual one as obtained after discovery. Within CHOReOS, this is done to consistently realize *separation of concerns*. That is, to separate pure coordination issues (i.e., undesired interactions) from adaptation ones (e.g., syntactic mismatches at the service interface level). The latter can arise whenever a service discovered as a participant does not exactly match the role to be played. Adaptation issues, as well as discovery ones, are out of scope for this paper and we refer to [4, 23] for details about their treatment within CHOReOS.

Structure of the paper – Section 2 introduces a *distributed social proximity network* scenario that is used as illustrative example. Section 3 formalizes the notion of CeFM. Its decomposition into a set of CMs is described in Section 4. Section 5 formalizes a distributed coordination algorithm, which describes the coordination logic that a CD has to perform by relying on its CM. We prove correctness of the distributed coordination algorithm in Section 6. To this extent, in Section 6, we also formalize the notion of undesired interactions. Section 7 discusses related works. Conclusions and final remarks are given in Section 8.

2 The distributed social proximity network scenario

With reference to Figure 2, choreography diagrams use rounded-corner boxes to denote choreography *tasks* (e.g., `getUserPref`). Each of them is labeled with the roles of the two participants involved in the task, and the name of the *service operation* performed by the initiating participant and provided by the other one. A role contained in a white box denotes the initiating participant (e.g., `IM`); a role contained in a filled box denotes the receiving participant (e.g., `UMS`).

The choreography in the figure models a small portion of a *Distributed Social Proximity Network* scenario[5], which concerns the collaboration of location-based services and citizen mobile apps. The choreography considers user preferences, user friend lists, and user location under predefined private data policy, to create ad-hoc itineraries between two citizens that know each other and are willing to meet in some place. In brief, upon receiving a start event from a citizen (i.e., from the dedicated mobile app named `SocialProxApp`), the Itinerary Manager (`IM`) calls the User Manager Service (`UMS`) to check if location sharing is enabled according to the citizen preferences. If yes (see the conditional branching

[5]Within CHOReOS, the scenario is part of a Dynamic Route system, which has been the pilot use case that we used to validate the overall choreography synthesis process.

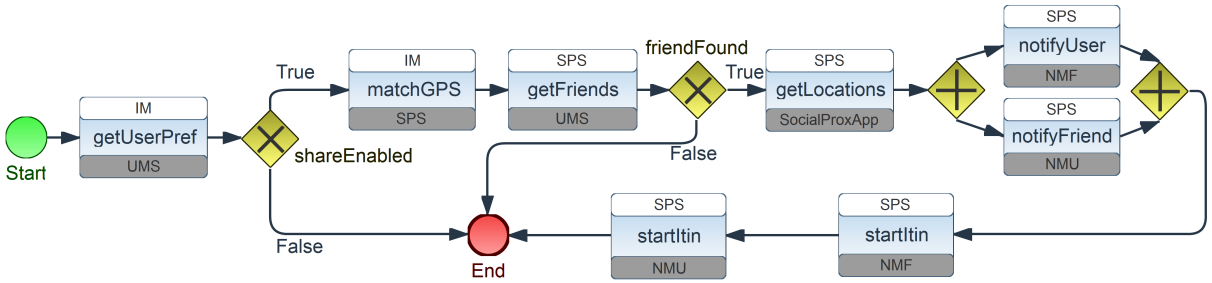

Figure 2: BPMN2 choreography diagram of the Distributed Social Proximity Network scenario

represented as a rhombus marked with a "x"), IM starts matching the GPS position of all citizen contacts by using the Social Proximity Service (SPS). Then, after interacting with UMS to get the list of all friends found nearby with location sharing enabled, SPS interacts with the SocialProxApps of the friends in the list to get their location. Finally, after a friend has been selected by the requesting citizen among the displayed list of available contacts, and the friend has accepted, SPS notifies in parallel the Notification Managers, NMU and NMF, for both the requesting user and the selected friend, respectively (see the parallel branch represented as an rhombus marked with a "+" with two outgoing arrows). These parallel flows are joined afterwards as soon as both the notification tasks have been accomplished (see the merging branch represented as an rhombus marked with a "+" with two incoming arrows), hence letting the two friends start their itineraries. An important aspect of this simple scenario is that the choreography in Figure 2 specifies that SPS must notify both the user and the friend before allowing them to start their respective itineraries. The choreography uses the merging branch at the right-hand side of the figure to model this desired behavior. When enforcing this choreography specification by using existing third-party services as participants, possible undesired interactions violating the specification can occur. For instance, the service playing the role of SPS attempts at notifying the user and starting his itinerary before notifying the friend.

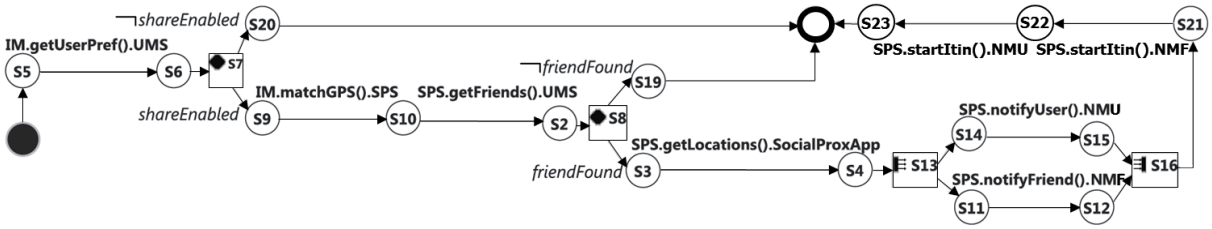

Figure 3: CeFM derived from the BPMN2 choreography diagram in Figure 2

Figure 3 shows the CeFM automatically derived out of the diagram in Figure 2. The numbering of the states in the CeFM depends on the performed model-based transformation. The symbol "¬" denotes the classical negation in propositional logic. Transitions are possibly labeled by the initiating participant, the performed task, and the receiving participant, separated by '.'.

In the following two sections, we formalize the notion of CeFM and its decomposition into a set of CMs, respectively.

3 Choreography explicit-Flow Model

Let $\mathscr{T}$ be the universal set of choreography tasks (e.g., service operations). Let $\mathscr{R}$ be the universal set of choreography roles. Let Φ be the universal set of Propositional Logic (PL) formulae. PL formulae

are used to distinguish choreography flows depending on whether a specific condition, represented as a *predicate* in PL, holds or not. More technically, according to the BPMN2 standard specification, in our implementation, a predicate is a conditional expression given in the syntax of the Java language. Thus, all data values involved in a predicate evaluation are assumed to actually range over finite domains.

The following definition characterizes the structural properties of a CeFM as required for enabling automated synthesis of CDs. As already introduced in Section 1, the intuition behind Definition 1 is that a BPMN2 choreography diagram has to be transformed into an equivalent choreography model that, within the CDs synthesis process, allows to statically infer coordination-related information.

Definition 1 (Choreography explicit-Flow Model) *A Choreography explicit-Flow Model (CeFM) C is a tuple* $(S_C^{ALT}, S_C^{LOOP}, S_C^{FORK}, S_C^{JOIN}, S_C^{PLAIN}, s_C^0, s_C^f, R_C, T_C, D_C)$*, where* $S_C^{ALT}, S_C^{LOOP}, S_C^{FORK}, S_C^{JOIN}$ *and* S_C^{PLAIN} *are disjoint and finite sets of states denoting* alternative, loop, fork, join, *and* plain *states, respectively.* s_C^0 *and* s_C^f *denote the* initial *(i.e., with no incoming transitions) and* final *(i.e., with no outgoing transitions) states, respectively, and they are neither alternative, loop, fork, join, nor plain states. Let* S_C *be the union set of all kinds of state, i.e.,* $S_C = S_C^{ALT} \cup S_C^{LOOP} \cup S_C^{FORK} \cup S_C^{JOIN} \cup S_C^{PLAIN} \cup \{s_C^0, s_C^f\}$*.* R_C *is a finite set of roles, i.e.,* $R_C \subseteq \mathscr{R}$*.* $T_C \subseteq (\mathscr{R} \times \mathscr{T} \times \mathscr{R}) \cup \Phi$ *is a set of task labels called the alphabet of C.* $D_C \subseteq S_C \times T_C \cup \{\varepsilon\} \times S_C$ *is the flow relation.* ε *denotes an* ε*-task (i.e., an "empty" task). C is finite if* D_C *is finite and C is empty if* D_C *is empty. We will make use of the following notations: (i)* $g \xrightarrow{\alpha}_C h$ *iff* $(g, \alpha, h) \in D_C \wedge \alpha = r.t.r'$ *for some* $r, r' \in R_C$*,* $t \in \mathscr{T}$*; (ii)* $g \xrightarrow{\rho}_C h$ *iff* $(g, \rho, h) \in D_C$ *for some* $\rho \in \Phi$*; and (iii)* $g \longrightarrow_C h$ *iff* $(g, \varepsilon, h) \in D_C$*. In all the cases we will write that g is a* predecessor *of h or, equivalently, h is a* successor *of g. The following structural properties hold:*

- **plain***: for all* $s^{plain} \in S_C^{PLAIN}$*,* s^{plain} *has only one predecessor* $s \in S_C \setminus \{s_C^f\}$ *such that* $s \longrightarrow_C s^{plain}$ *and* s^{plain} *has only one successor* $s' \in S_C \setminus \{s_C^0\}$ *such that* $s^{plain} \longrightarrow_C s'$*;*
- **alternative:** *for all* $s^{alt} \in S_C^{ALT}$*,* s^{alt} *has only one predecessor* $s \in S_C \setminus \{s_C^f\}$ *such that* $s \longrightarrow_C s^{alt}$ *and* s^{alt} *has at least two successors* $s_1, s_2 \in S_C \setminus \{s_C^0\}$ *such that* $s^{alt} \xrightarrow{\rho_1}_C s_1$ *and* $s^{alt} \xrightarrow{\rho_2}_C s_2$ *for some* $\rho_1, \rho_2 \in \Phi$*; for all pairs of successors* (s_i', s_j')*, with* $i \neq j$*, and respective predicates* $\rho_i, \rho_j \in \Phi$ *such that* $s^{alt} \xrightarrow{\rho_i}_C s_i'$ *and* $s^{alt} \xrightarrow{\rho_j}_C s_j'$ *then there is no variable assignment that makes* $\rho_i \wedge \rho_j$ *hold;*
- **loop:** *for all* $s^{loop} \in S_C^{LOOP}$*,* s^{loop} *has only two predecessors* $s, s' \in S_C \setminus \{s_C^f\}$ *such that* $s \longrightarrow_C s^{loop}$ *and* $s' \longrightarrow_C s^{loop}$*, and* s^{loop} *has only two successors* $q, q' \in S_C \setminus \{s_C^0\}$ *such that* $s^{loop} \longrightarrow_C q$*,* $s^{loop} \xrightarrow{\rho}_C q'$ *for some* $\rho \in \Phi$*, and* s' *is reachable from* q'*; we call* q' *and q the* loop entry state *and* exit state*, respectively;*
- **fork:** *for all* $s^{fork} \in S_C^{FORK}$*,* s^{fork} *has only one predecessor* $s \in S_C \setminus \{s_C^f\}$ *such that* $s \longrightarrow_C s^{fork}$ *and* s^{fork} *has at least two successors* $s', s'' \in S_C \setminus \{s_C^0\}$ *such that* $s^{fork} \longrightarrow_C s'$ *and* $s^{fork} \longrightarrow_C s''$*;*
- **join:** *for all* $s^{join} \in S_C^{JOIN}$*,* s^{join} *has only one successor* $s \in S_C \setminus \{s_C^0\}$ *such that* $s^{join} \longrightarrow_C s$ *and* s^{join} *has at least two predecessors* $s', s'' \in S_C \setminus \{s_C^f\}$ *such that* $s' \longrightarrow_C s^{join}$ *and* $s'' \longrightarrow_C s^{join}$*.*

BPMN2 specification employs the theoretical concept of *token* that, traversing the sequence flows and passing through the elements in a BPMN2 process, aids to define its behavior. In line with this concept, the formal semantics of a CeFM is given in Section 4 as a set of coordination models (see Definition 3) obtained through the automatic generation process formalized by Definition 4. In particular, the generation process leverages the notion of *structural sequential flow* in Definition 2 to formally model all the paths traversing the choreography diagram. The coordination models are meant to coordinate these paths according to the different state types.

Intuitively, the initial state of a CeFM generates the token that must eventually be consumed at the final state. If a token is on a plain state, then the outgoing flow is ready to be executed hence performing the operation characterized by its task label. If a token is on an alternative state, then one of the outgoing flows whose predicate holds is ready to be executed. This means that, according to the above structural properties, we deal with BPMN2 Diverging (Decision) Exclusive Gateways through alternative states whose outgoing flows are guarded by mutually exclusive predicates, i.e., at run time only one predicate evaluates to true[6]. At the same time, according to the BPMN2 standard specification, we disallow the specification of non deterministic gateways with no condition specified. If a token is on a loop state, then the outgoing flow leading to the loop entry state is ready to be executed if its guarding predicate holds. Otherwise, the flow leading to the exit state will be performed. If a token is on a fork state, then all outgoing flows are taken simultaneously leading to several parallel executions, each having its own token. If, for each of its incoming flows, there is a token on a join state, then the outgoing flow is triggered.

Due to the introduction of ε-tasks, in general, a CeFM is non-deterministic. ε-tasks are introduced for dealing with non-plain states both structurally and semantically. For instance, they can be used to specify cascading fork states or to model the (internal) event of creating parallel flows, which originate from a fork state.

By leveraging the concept of token, alternative models, such as free-choice Petri nets [16], might have been adopted. However, the deep study we have initially conducted within CHOReOS to precisely define, at the project level, the integration architecture for the CHOReOS Integrated Development and Run-time Environment[7] (IDRE), led the whole consortium to agree on the definition of the CeFM model, which best met the (both formal and technical) requirements of all the software tools now integrated by the IDRE. Indeed, to the purposes of defining an integrated suite of tools to support the whole choreography life cycle, the CeFM model brings together many features of already existing formalisms and notations in the literature, and filters out those ones not strictly needed. Last but not least, one of the main requirements was to have a notation as close as possible to the BPMN2 choreography diagrams, while enabling formal reasoning and fully automatic treatment by all the IDRE components.

4 Automated decomposition of a CeFM into a set of CMs

From hereon let $C=(S_C^{ALT},S_C^{LOOP},S_C^{FORK},S_C^{JOIN},S_C^{PLAIN},s_C^0,s_C^f,R_C,T_C,D_C)$ be a CeFM. The ability of explicitly identifying specific structural sequential flows of C represents a basic ingredient to support the automated synthesis of the CDs that cooperatively realize C. For instance, in order to deal with the synchronization of several parallel flows in correspondence of a join state, it is sufficient for a CD to extract out of C each of these flows as originating from the same fork. This allows a CD, reaching a join state, to be aware of (i) those CDs that must be notified about the reach of the join state and, complementary, (ii) those states (predecessors of the join state) that must be waited for.

Definition 2 (Structural sequential flow) *The sequence $t=\alpha_{i+1}\alpha_{i+2}\ldots\alpha_n$ is a* structural sequential flow *of C iff there exists a sequence of states $s_i\ldots s_n \in S_C$ such that $s_i \xrightarrow{\alpha_{i+1}}_C s_{i+1}\ldots s_{n-1}\xrightarrow{\alpha_n} s_n$, $i \geq 0$, $n > i$. The* empty flow *is denoted by $\ddot{\varepsilon}$. We will make use of the following notation: $s_1 \overset{\alpha}{\Longrightarrow}_C s_n$ iff there exists $n \geq 2$ such that $s_1 \xrightarrow{\alpha}_C s_2 \longrightarrow_C \ldots \longrightarrow_C s_n$.*

As introduced in Section 1, a further basic ingredient of our synthesis method concerns the distribution step for C. That is, C is distributed into a set of CMs, one for each CD. We recall that CDs are not

[6] Indeed, in our implementation we also allow the specification of default a branch, not considered here for simplicity.

[7] `http://www.choreos.eu/bin/view/Documentation/WebHome`

always generated for each pair of participants. Indeed, for two given participants, a CD is generated only when there is a dependency relation between them. This means that we do not require to always keep the global state of the choreography to enforce it. This characterizes the distributed nature of our synthesis approach.

Each CM represents a local view of C. It is local since it models the coordination logic that has to be enforced on the interaction between two participants, p_i and p_j. This is done by exchanging coordination information among the CD supervising p_i and the CDs it needs to synchronize with. Thus, the set of all CMs can be considered as a distributed model of C. A CM characterizes the coordination information that is needed to automatically synthesize the coordination logic that the corresponding CD has to perform, while interacting with the other CDs, in order to realize C.

Definition 3 (Coordination Model) *Let $i,j \in \mathbb{N}$, with $i \neq j$, be the identifiers of two participant services. Then, the* Coordination Model *$CM_{i,j}$ for the operations required by the participant i and provided by the participant j is the set[8] of tuples $\{\tau \mid \tau \in S_C \times T_C \times S_C \times 2^{\mathbb{N}\times\mathbb{N}} \times \Phi \times 2^{S_C\times\mathbb{N}\times\mathbb{N}} \times 2^{S_C\times\mathbb{N}\times\mathbb{N}}\}$. For each tuple $\langle s,t,s',CD_{s'},\rho,Notify_s,Wait_{siblings(s)}\rangle$:*

- *s denotes the CeFM source state from which the related CD can either perform the operation t or take a move without performing any operation (i.e., the CD can step over an ε-task). In both cases, s' denotes the reached target state;*
- *$CD_{s'}$ contains the set of (identifiers of) those CDs whose supervised services became active in s', i.e., the ones that will be allowed to require/provide some operation from s'. This information is used by the "currently active" CDs to inform the set of "to be activated" CDs (in the target state) about the changing global state;*
- *ρ is a PL predicate whose validity has to be checked to select the correct tuple, and hence the correct flow(s) in the CeFM;*
- *$Notify_s$ contains the predecessor of a join state that a CD, when reaching it, must notify to the other CDs in the parallel flow(s) of the same originating fork. Complementary, $Wait_{siblings(s)}$ contains the predecessors of join states that must be waited for.*

According to the above definition, each $CM_{i,j}$ is automatically synthesized out of C as formalized by the following definition.

Definition 4 (Coordination Model Generation) *Let $1,\ldots,n$ be identifiers for n participant services playing the roles $r_1,\ldots,r_n \in \mathscr{R}$, respectively. Then, $CM_{i,j}$* is generated out of *C as follows:*

$$CM_{i,j}=\{\langle s,t,s',CD_{s'},true,Notify_s,Wait_{siblings(s)}\rangle \mid s \xrightarrow{r_i.t.r_j}_C s'\} \cup$$
$$\{\langle s,\varepsilon,s',CD_{s'},true,Notify_s,Wait_{siblings(s)}\rangle \mid \exists s^{prec},t\colon s^{prec} \xRightarrow{r_i.t.r_j}_C s \longrightarrow_C s'\} \cup$$
$$\{\langle s,\varepsilon,s',CD_{s'},\rho_{s.s'},Notify_s,Wait_{siblings(s)}\rangle \mid \exists s^{prec},t\colon s^{prec} \xRightarrow{r_i.t.r_j}_C s \xrightarrow{\rho_{s.s'}}_C s'\}$$

where

$$CD_{s'}=\{(h,k) \mid (h,k)\neq(i,j) \wedge \exists s'',t\colon s' \xrightarrow{r_h.t.r_k}_C s''\} \cup$$
$$\{(h,k) \mid (h,k)\neq(i,j) \wedge \exists \rho\in\Phi, s'',s^{succ},t\colon s' \xrightarrow{\rho}_C s'' \xrightarrow{r_h.t.r_k}_C s^{succ}\} \cup$$
$$\{(h,k) \mid (h,k)\neq(i,j) \wedge s'\in S_C^{FORK}\cup S_C^{JOIN} \wedge \exists s'',s^{succ},t\colon s' \longrightarrow_C s'' \xRightarrow{r_h.t.r_k} s^{succ}\}$$
$$Notify_s=\{[s,(h,k)] \mid s'\in S_C^{JOIN} \wedge \exists s''\neq s, s^{prec},t\colon s^{prec} \xRightarrow{r_h.t.r_k}_C s'' \longrightarrow_C s'\}$$
$$Wait_{siblings(s)}=\{[s'',(h,k)] \mid s'\in S_C^{JOIN} \wedge \exists s''\neq s, s^{prec},t\colon s^{prec} \xRightarrow{r_h.t.r_k}_C s'' \longrightarrow_C s'\}$$

$\mathbf{CM_{IM,UMS}}$	$\mathbf{CM_{SPS,UMS}}$
$\langle S5, getUserPref(), S6, \{\}, true, \{\}, \{\}\rangle$ $\langle S6, \varepsilon, S7, \{\}, true, \{\}, \{\}\rangle$ $\langle S7, \varepsilon, S20, \{\}, \neg shareEnabled, \{\}, \{\}\rangle$ $\langle S7, \varepsilon, S9, \{(IM, SPS)\}, shareEnabled, \{\}, \{\}\rangle$ $\langle S20, \varepsilon, Final, \{\}, true, \{\}, \{\}\rangle$	$\langle S10, getFriends(), S2, \{\}, true, \{\}, \{\}\rangle$ $\langle S2, \varepsilon, S8, \{\}, true, \{\}, \{\}\rangle$ $\langle S8, \varepsilon, S19, \{\}, \neg friendFound, \{\}, \{\}\rangle$ $\langle S8, \varepsilon, S3, \{(SPS, SocialProxApp)\}, friendFound, \{\}, \{\}\rangle$ $\langle S19, \varepsilon, Final, \{\}, true, \{\}, \{\}\rangle$
$\mathbf{CM_{IM,SPS}}$	$\mathbf{CM_{SPS,SocialProxApp}}$
$\langle S9, matchGPS(), S10, \{(SPS, UMS)\}, true, \{\}, \{\}\rangle$	$\langle S3, getLocations(), S4, \{\}, true, \{\}, \{\}\rangle$ $\langle S4, \varepsilon, S13, \{\}, true, \{\}, \{\}\rangle$ $\langle S13, \varepsilon, S11, \{(SPS, NMF)\}, true, \{\}, \{\}\rangle$ $\langle S13, \varepsilon, S14, \{(SPS, NMU)\}, true, \{\}, \{\}\rangle$
$\mathbf{CM_{SPS,NMU}}$	$\mathbf{CM_{SPS,NMF}}$
$\langle S14, notifyUser(), S15, \{\}, true, \{\}, \{\}\rangle$ $\langle S15, \varepsilon, S16, \{\}, true, \{[S15, (SPS, NMF)]\},$ $\{[S12, (SPS, NMF)]\}\rangle$ $\langle S16, \varepsilon, S21, \{(SPS, NMF)\}, true, \{\}, \{\}\rangle$ $\langle S22, startItin(), S23, \{\}, true, \{\}, \{\}\rangle$ $\langle S23, \varepsilon, Final, \{\}, true, \{\}, \{\}\rangle$	$\langle S11, notifyFriend(), S12, \{\}, true, \{\}, \{\}\rangle$ $\langle S12, \varepsilon, S16, \{\}, true, \{[S12, (SPS, NMU)]\},$ $\{[S15, (SPS, NMU)]\}\rangle$ $\langle S16, \varepsilon, S21, \{\}, true, \{\}, \{\}\rangle$ $\langle S21, startItin, S22, \{(SPS, NMU)\}, true, \{\}, \{\}\rangle$

Table 1: Coordination Models Tuples

Going back to our illustrative example, Table 1 reports the CMs derived from the CeFM in Figure 3. Intuitively, the first tuple in $CM_{IM,UMS}$ specifies that $CD_{IM,UMS}$ can perform the operation $getUserPref$ from the source state $S5$ to the target state $S6$; whereas, the second tuple specifies that $CD_{IM,UMS}$ can step over $S6$ and reach $S7$, from where alternative branches can be undertaken. Then, as specified by the third and fourth tuple, $CD_{IM,UMS}$ can reach either $S20$ or $S9$ according to the evaluation of the related conditions, i.e., $\neg shareEnabled$ or $shareEnabled$, respectively. This means that, after $getUserPref$ has been requested by IM and forwarded to UMS, in the case $shareEnabled$ holds, $CD_{IM,UMS}$ uses the fourth tuple $\langle S7, \varepsilon, S9, \{(IM, SPS)\}, shareEnabled, \{\}, \{\}\rangle$ to step over the alternative state $S7$, reach $S9$, and inform $CD_{IM,SPS}$ about the new global state $S9$.

Considering the second tuple $\langle S15, \varepsilon, S16, \{\}, true, \{[S15, (SPS, NMF)]\}, \{[S12, (SPS, NMF)]\}\rangle$ in $CM_{SPS,NMU}$, $CD_{SPS,NMU}$ notifies $S15$ to $CD_{SPS,NMF}$ and waits for receiving the notification by $CD_{SPS,NMF}$ about $S12$. On the other hand, considering the tuple $\langle S12, \varepsilon, S16, \{\}, true, \{[S12, (SPS, NMU)]\}, \{[S15, (SPS, NMU)]\}\rangle$ in $CM_{SPS,NMF}$, $CD_{SPS,NMF}$ notifies $S12$ to $CD_{SPS,NMU}$ and waits for receiving the notification by $CD_{SPS,NMU}$ about $S15$. Note that the same considerations apply in case different parallel threads of the same CD are involved in a join state.

5 Choreography enforcement via distributed coordination

In this section we formalize the distributed coordination algorithm that describes the coordination logic that each CD has to perform by relying on its CM. The algorithm inherits the distributed mutual exclusion principle and leverages some foundational notions (such as happened-before relation, partial ordering, and time-stamps) of the algorithm in [29]. More precisely, the enforcement of mutual exclusive access to a single resource in [29] is scaled up to the enforcement of concurrent task flows according to arbitrarily complex choreographies. In the style of [29], the most appropriate way to present the algorithm is to define rules that each delegate $CD_{i,j}$ follows in a distributed setting, when its supervised service S_i sends a request to perform a task t with S_j (without relying on any central synchronizing entity or shared memory). The actions defined by each rule are assumed to form a single event (i.e., each rule is atomic). As detailed below, a CD is a *reactive* entity that, at each iteration of the algorithm, WAITs for receiving either a request from the service it supervise or NOTIFY/UPDATE messages from the other CDs.

[8] As usual, let S be a set, 2^S denotes the power-set of S

The rules locally characterize the collaborative behavior of the CDs at run-time from a clear *one-to-many* point of view. The most important aspect here is that, upon reaching a join state, the *involved* CDs notify and wait for each other in order to synchronize their execution. At run time, NOTIFY messages (together with a simple notion of *priority* $P_{i,j}$, initially associated to each $CD_{i,j}$) are used to realize join states as distributed synchronization points. Then, when a join state has been realized, and hence all the parallel executions have been synchronized, the CD with highest priority is in charge of notifying the CDs that, according to the choreography, are allowed to proceed. Each $CD_{i,j}$ sends NOTIFY messages according to the coordination information contained by the sixth element of the $CM_{i,j}$ tuples. The "co-related" WAIT primitive is instead performed according to the information contained in the seventh element of the tuples. In our implementation, the correct co-relation among WAIT primitives and NOTIFY (or UPDATE) messages is realized by means of dedicated queues that, for each WAIT, buffer the expected NOTIFYs (or UPDATEs). $CD_{i,j}$ uses UPDATE messages whenever the current global state of C changes according to the performed coordination. By considering the coordination information represented by the fourth element of its tuples, $CD_{i,j}$ sends an UPDATE message in order to inform, about the state change, those CDs whose execution can progress from the new current global state. Thus, if a $CD_{h,k}$ receives a request to perform a task t while its local execution is in a state where t is not allowed, then it waits for receiving an UPDATE message on a state from which t is allowed.

In Table 2, we defined three rules. In brief, Rule 1 governs the exchange of business-level messages by considering the cases in which: a CD receives a message that is allowed by C, hence forwarding it to the interested service as soon as the CD reaches the enabling state of C; and a CD steps over ε-tasks by exploiting the procedure defined in Table 3. Leveraging the distribution step of Definition 4 (which is performed statically), the procedure is capable to handle in a uniform way all kinds of states, i.e., plain, fork, join, alternative, and loop states. At coordination-level, Rule 2 and Rule 3 regulate the exchange of the UPDATE and NOTIFY messages, respectively. Specifically, Rule 2 updates the local current state of the CD according to the current global state of C as received through the UPDATE message. Rule 3 allows the CD to proceed only after all the needed NOTIFY messages have been received. For the sake of presentation, the algorithm assumes that each alternative branch has at least one (and only one) enabling condition that evaluates to true. Furthermore, each participant does not request to perform a task that is not present in the choreography specification. However, our implementation of the algorithm is able to detect these cases raising specific exceptions.

Adhering to Definition 4, the algorithm takes as input $CM_{i,j}$. Thus, let τ be a tuple in $CM_{i,j}$:

- τ[`src`] (resp., τ[`trg`]) is the enabling source (resp., target) state of the transition labeled with τ[`allowedOp`];
- τ[`allowedOp`] is the operation that can be performed by S_i when in the source state;
- τ[`allowedService`] is the set of CDs (and hence, of supervised services) that can proceed from the target state;
- τ[`cond`] is the PL predicate associated to an alternative path, which allows a CD to proceed on that path whenever it holds;
- τ[`toBeNotified`] (resp., τ[`toBeWaited`]) is the set of CDs that, progressing on parallel paths together with $CD_{i,j}$, must be notified (resp., waited for) as soon as $CD_{i,j}$ reaches the state joining the paths.

At the beginning, all CDs are waiting for receiving an initiating UPDATE message to internally set the state(s) from which they can perform an operation. These messages are sent by an activating compo-

nent, specifically developed within CHOReOS, called Enactment Engine (`https://github.com/choreos/enactment_engine`). The latter is also in charge of automatically deploying the CDs.

Rule 1: Upon receiving, in the current state s of C, a request from S_i to perform a task t with S_j,

1.1 if there exists $\tau \in CM_{i,j}$ *s.t.* $\tau[\texttt{src}] = s$ and $\tau[\texttt{allowedOp}] = t$ (i.e., t is allowed from s) **then**

1.1.1 $CD_{i,j}$ forwards to S_j the message initiating t, and gives the control back to S_i [9];

1.1.2 $CD_{i,j}$ updates s to $\tau[\texttt{trg}]$;

1.1.3 for all $(h,k) \in \tau[\texttt{allowedService}]$, $CD_{i,j}$ sends UPDATE($\tau[\texttt{trg}]$) to $CD_{h,k}$;

1.1.4 **`StepOver`**(s);

else if $\tau[\texttt{src}] \neq s$ for each $\tau \in CM_{i,j}$ *s.t.* $\tau[\texttt{allowedOp}] = t$ (i.e., t is not allowed from s) **then**

1.1.5 $CD_{i,j}$ WAITs for receiving UPDATE($\tau[\texttt{src}]$), hence temporarily blocking S_i [10]

Rule 2: When $CD_{i,j}$ receives UPDATE(*state*) for some *state* that it was waiting for **then**

2.1 $CD_{i,j}$ updates s to *state*;

2.2 if a task t is pending [11] **then goto** Rule 1.1;

Rule 3: When $CD_{i,j}$ receives all NOTIFY(*predecessor*, $CD_{h,k}$, *join*) it was waiting for[12] **then**

3.1 if $CD_{i,j}$ has the highest priority (i.e., for all $CD_{h,k}$, $P_{i,j} > P_{h,k}$) **then**

3.1.1 $CD_{i,j}$ updates s to *join*;

3.1.2 let $\tau \in CM_{i,j}$ be *s.t.* $\tau[\texttt{src}] = join$;

3.1.3 for all $(h,k) \in \tau[\texttt{allowedService}]$, $CD_{i,j}$ sends UPDATE($\tau[\texttt{trg}]$) to $CD_{h,k}$;

3.2 if a task t is pending **then goto** Rule 1.1;

Table 2: Rule-based description of the algorithm

`StepOver`(s):

while there exists $\tau \in CM_{i,j}$ *s.t.* $\tau[\texttt{src}] = s$ and $\tau[\texttt{allowedOp}] = \tau.\varepsilon.\tau$ (i.e., it is possible to proceed from s only by an internal task) and there exists $\tau' \in CM_{i,j}$ *s.t.* $\tau'[\texttt{src}] = \tau[\texttt{trg}]$ and $\tau'[\texttt{cond}]$ holds **do**

`StepOver`($\tau'[\texttt{trg}]$);

for all $[s,(h,k)] \in \tau[\texttt{toBeNotified}]$, $CD_{i,j}$ sends NOTIFY(s, $CD_{h,k}$, $\tau[\texttt{trg}]$) to $CD_{h,k}$;

for all $[s'',(h,k)] \in \tau[\texttt{toBeWaited}]$, $CD_{i,j}$ WAITs for receiving NOTIFY(s'', $CD_{h,k}$, $\tau[\texttt{trg}]$) from $CD_{h,k}$, hence temporarily blocking S_i (see Rule 3);

for all $(h,k) \in \tau'[\texttt{allowedService}]$, $CD_{i,j}$ sends UPDATE($\tau'[\texttt{trg}]$) to $CD_{h,k}$;

Table 3: **`StepOver`** procedure

[9] Note that (*i*) the Service-CD interaction is synchronous; (*ii*) the CD-CD interaction is either synchronous or asynchronous; (*iii*) the CD-CD exchange of coordination information is asynchronous.

[10] See Rule 2. In other words, $CD_{i,j}$ is in a source state, say s', different from s and it is waiting for being notified for s' to become the new current state of C. Note that, since the service-to-CD interaction is synchronous, the task t is pending and S_i is waiting for receiving the control back from $CD_{i,j}$.

[11] That is, upon receiving a request to perform t, $CD_{i,j}$ was waiting to receive UPDATE(*state*).

[12] See Rule 1.2.4.

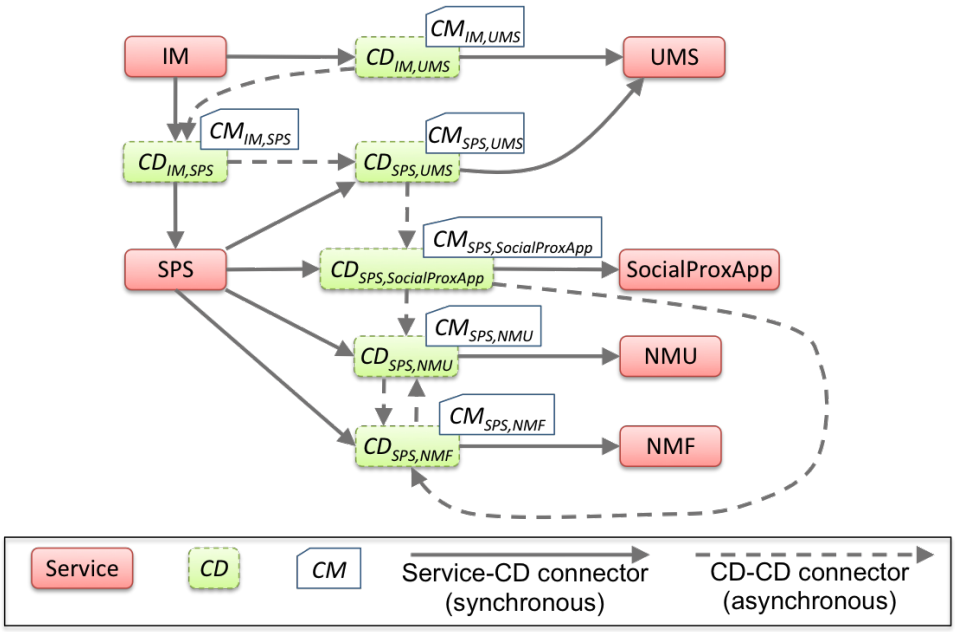

Figure 4: Architecture of the enforced Distributed Social Proximity Network scenario

Going back to our illustrative example, Figure 4 shows the software architecture of the system that realizes the choreography specified by the CeFM shown in Figure 3. This system is composed of the services that instantiate the choreography participant roles (e.g., `IM`, `SPS`), the automatically synthesized CDs (e.g., $CD_{IM,SPS}$, $CD_{SPS,UMS}$) together with their CMs (e.g., $CM_{IM,SPS}$, $CM_{SPS,UMS}$), synchronous connectors enabling Service-CD interaction by exchanging business level messages (e.g., the *getUserPref* and *notifyUser* operations), and asynchronous connectors enabling CD-CD interaction by exchanging additional coordination messages (e.g., NOTIFY and UPDATE messages). The deployment information needed to realize the structure of the depicted architecture is automatically synthesized by exploiting the set of generated CMs.

As introduced above, at the beginning, $CD_{IM,SPS}$ is waiting for receiving an UPDATE on state $S9$ (of the CeFM); $CD_{IM,UMS}$ is waiting for receiving an UPDATE on $S5$; $CD_{SPS,UMS}$ is waiting for receiving an UPDATE on $S10$; $CD_{SPS,SocialProxApp}$ is waiting for receiving an UPDATE on $S3$; $CD_{SPS,NMU}$ is waiting for receiving an UPDATE on either $S14$ or $S22$; and $CD_{SPS,NMF}$ is waiting for receiving an UPDATE on either $S11$ or $S21$. Again, this information is automatically synthesized out of the generated CMs. Once the synthesized CDs are deployed, by obeying the constraints of the architecture shown in Figure 4, the CHOReOS Enactment Engine sends UPDATE($S5$) to $CD_{IM,UMS}$, hence starting the execution of the choreography.

To illustrate the execution of the distributed coordination algorithm at work on our example, Figure 5 shows a sequence diagram representing an excerpt[13] of the exchange of business- and coordination-level messages among the participants and their supervising CDs. As it is evident from the diagram, the collaboration of the synthesized CDs coordinates the interaction among the participant services in order to let them perform only the interactions specified by the CeFM, hence preventing, e.g., the undesired interaction mentioned in Section 2. We recall that this undesired interaction concerns the case in which a notified friend takes on the created itinerary when the other friend has not been notified yet.

[13]From the initial state to S21, and by assuming that both *shareEnabled* and *friendFound* hold.

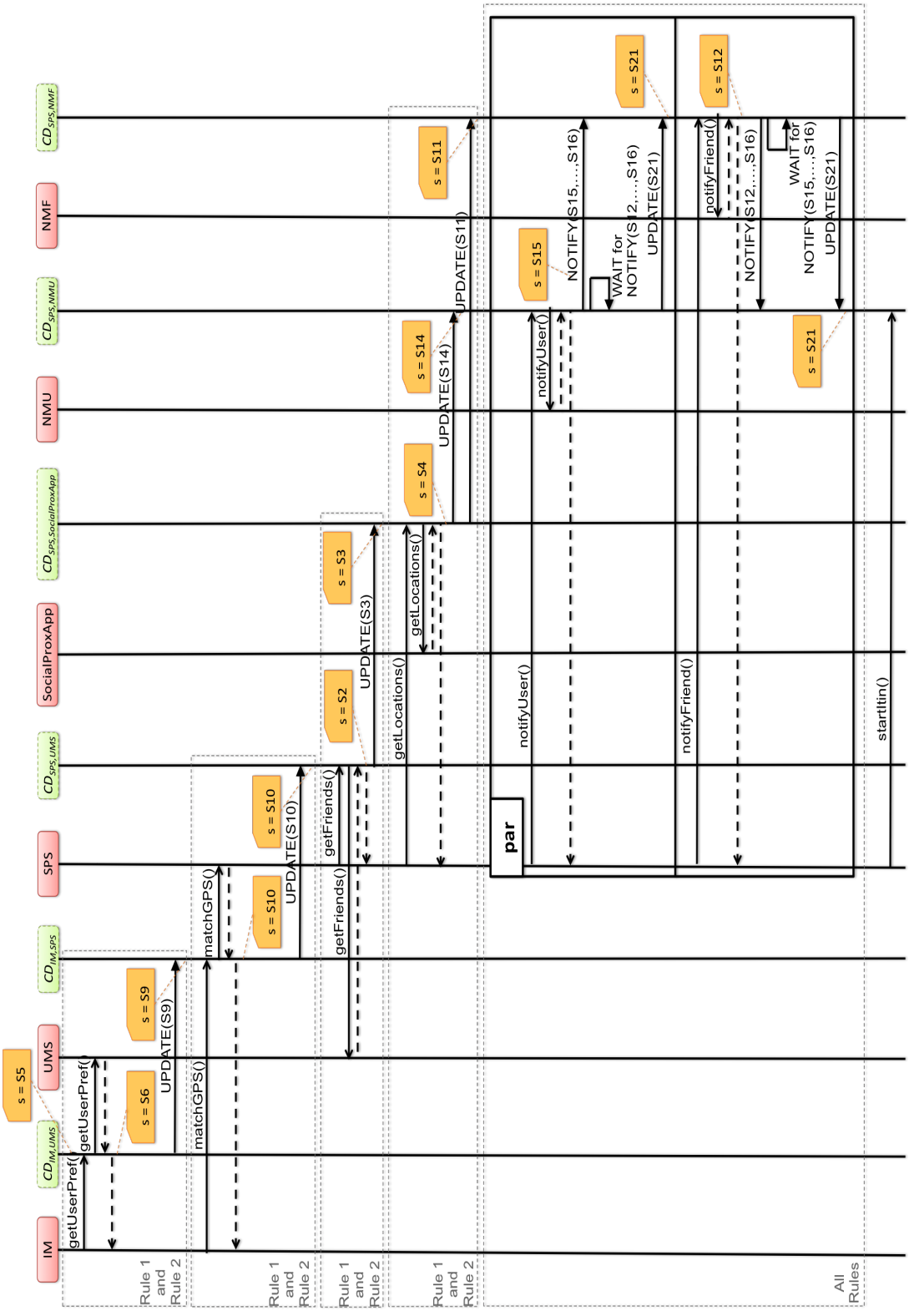

Figure 5: Messages exchange for the Distributed Social Proximity Network scenario

6 Correctness

In this section, we give a rigorous characterization of undesired interactions, and prove that our enforcement method prevents them.

For our distributed coordination algorithm to be correct, it should never happen that:

(i) a CD accepts a request to perform a task that is not allowed in the current state;

(ii) although the task is allowed, the current state (a) has been reached by an alternative state whose condition does not hold; (b) is a loop entry state but the guard of the loop does not hold; (c) is a loop exit state but the guard still holds; (d) is the successor of a join state but some parallel flows must still be completed; (e) leads to a predecessor of a join state where, after performing the requested task, each involved CD is waiting for the others.

These considerations lead to the following definition that characterizes the way undesired interactions can be detected at run-time. That is, an undesired interaction occurs whenever an *undesired operation*, as defined by Definition 5, occurs. In the definition, we make use of the following notations. $CM_{i,j}(s)$ denotes the set of tuples in $CM_{i,j}$ with s as first element. $CM_{i,j}(s)\restriction_{Op}$ denotes the set of choreography operations (different from ε) appearing as second element of tuples in $CM_{i,j}(s)$ (i.e., all the operations allowed from s). $CM_{i,j}(s)\restriction_{Wait}$ denotes the union of valid triples in $Wait_{siblings(s)}$ (i.e., all the predecessors of s that must be waited for at run-time). The predicate $\sim Notify_{i,j}$ holds if and only if $CD_{i,j}$ has not received yet all NOTIFY messages that it is waiting for.

Definition 5 (Undesired operation) *Let $CM_{i,j}$ be the CM generated for the delegate $CD_{i,j}$ out of the CeFM. Let $s \in S_C$ be the current state reached by $CD_{i,j}$ during its execution. A choreography operation $\alpha = r_i.t.r_j$ (for some $r_i, r_j \in R_C$ and $t \in \mathscr{T}$) is an* undesired operation *in s if and only if one of the following conditions hold:*

- **undesired task:** $\alpha \notin CM_{i,j}(s)\restriction_{Op}$;
- **undesired flow:** $\alpha \in CM_{i,j}(s)\restriction_{Op} \wedge$ *one of the following conditions hold:*
  - **invalid alternative:** $\exists s^{alt} \in S_C^{ALT}, \rho \in \Phi : s^{alt} \xrightarrow{\rho}_C s \wedge \neg\rho$;
  - **invalid loop:** $\exists s^{loop} \in S_C^{LOOP}, \rho \in \Phi : s^{loop} \xrightarrow{\rho}_C s \wedge \neg\rho$;
  - **missed loop:** $\exists s^{loop} \in S_C^{LOOP}, \rho \in \Phi, s' : s^{loop} \longrightarrow_C s \wedge s^{loop} \xrightarrow{\rho}_C s' \wedge \rho$;
  - **missed join:** $\exists s^{join} \in S_C^{JOIN} : s^{join} \longrightarrow_C s \wedge \sim Notify_{i,j}$;
  - **deadlocking join:** $\exists s^{join} \in S_C^{JOIN}, s' : s \xrightarrow{\alpha}_C s' \longrightarrow s^{join} \wedge CD_{i,j}$ *is in* $s' \wedge \sim Notify_{i,j} \wedge \forall (s'', h, k) \in CM_{i,j}(s)\restriction_{Wait} : CD_{h,k}$ *is in* $s'' \wedge \sim Notify_{h,k}$.

Correctness proof (by contradiction) – Let us suppose that, at run-time, the collaboration of the synthesized CDs and the participants performs an undesired operation. By Definition 5, this means that either an undesired task or an undesired flow has been performed. By referring to the distributed coordination algorithm defined in Section 5, the **undesired task** condition contradicts **Rule 1.1**. Now, let us consider all the cases that characterize the **undesired flow** condition. The **invalid alternative**, **invalid loop**, **missed loop**, and **missed join** conditions contradict the execution of procedure `StepOver`; finally, the **deadlock** condition contradicts the combined execution of `StepOver` with **Rule 3** and **Rule 2**. In any case, we deduce a contradiction and, hence, the correctness proof is given meaning that the interaction among the generated CDs and the participants prevents any kind of undesired interaction.

7 Related work

The method formalized in this paper is related to a large number of other approaches that have been considered in the literature in the domains of *service-oriented and component-based engineering*, and *controller synthesis and priority scheduling*. We discuss here only the ones that are closer to the automated choreography enforcement problem.

Service-oriented and component-based engineering – There are many approaches aiming at composing services by means of BPEL, WSCI, or WS-CDL choreographers [10, 11, 30, 31, 33, 37, 40]. The common idea underlying these approaches is to assume a high-level specification of the requirements that the choreography has to fulfil and a behavioural specification of the participants. From these two assumptions, by applying data and control-flow analysis, the BPEL, WSCI or WS-CDL description of a centralized choreographer is automatically derived so that it satisfies the specified requirements. In particular, in [40], the authors propose an approach to derive service implementations from a choreography specification. In [37], the authors assume that some services are reused and propose an approach to exploit wrappers to make the reused services match the choreography. Most of the previous approaches concern orchestration, which is a form of composition different form choreography. The former focuses on centralized coordination, hence resulting in a monolithic composition. Instead, the latter is a means for composing services in a fully distributed way.

Despite the fact that the works described in [37, 40, 7, 22, 8] focus on choreography, they consider the problem of checking choreography realizability. Note that it is a fundamentally different problem from the one considered in this paper. In fact, our approach is reuse-oriented and aims at restricting, by means of the synthesized CDs, the interaction behaviour of the discovered (third-party) services in order to realize the specified choreography. Differently, the approaches described in [37, 40, 7, 22, 8] are focused on verifying whether the set of services, required to realize a given choreography, can be easily implemented by simply considering the role-based local views of the specified choreography. That is, this verification does not aim at synthesizing the coordination logic, which is needed whenever the collaboration among the participants leads to global interactions that violate the choreography behaviour.

Similarly, later work in [17, 35] valuably advances the state-of-the-art results. Then, most closely to our work, in [20] the authors presents a framework for verifying choreographies using model- and equivalence-checking techniques. The framework enables the verification of some analysis tasks, i.e., repairability, realizability, conformance, synchronizability, and control for enforcing realizability. In order to check in sequence the system synchronizability and realizability using equivalence checking, distributed controllers are generated through an iterative process presented in [21]. Counterexamples are exploited to refine the behaviour of the controllers by adding new synchronization messages until both synchronizability and realizability can be enforced.

Works in the area of the synthesis of runtime monitors from automata are described in [38, 39]. Note that runtime monitoring is mostly focused on the detection of undesired behaviours, while runtime enforcement focuses on their prevention.

In the context of *Reo connectors*, a number of related works are worth to be discussed. The works described in [27, 28, 2] discuss different approaches to extract coordination/choreography specifications from BPMN diagrams, UML state diagrams, and UML activity diagrams. Automated synthesis of coordination/choreography specifications from scenario-based specification is addressed in [32, 1]. The works described in [25, 24, 26] are more specific with respect to the problem of distributed enforcement/implementation of choreographies. In particular, in [25], the authors provide a systematic and rigorous way of constructing hybrid (i.e., partially-distributed) connector implementations for distributed orches-

trations. In [24], the authors define a new product operator for *Constraint Automata* (CA) whose computation at run-time requires only relatively simple distributed algorithms. The work in [26] describes an hybrid approach for the automatic code generation of orchestrations with Reo. The main focus of these works is on hybrid enforcement approaches.

Controller synthesis and priority scheduling – The controller synthesis problem is conceptually similar to ours (yet technically different). That is, given a system specification and a property, synthesize a controller that, once put in parallel with the system, enforces the satisfaction of the property. In [5], the authors describe a compositional approach for efficiently constructing a centralized controller. Then, in [15], they describe how to decompose a monolithic controller into a network of Reo connectors. The former focuses on compositionality of the synthesis rather than on controller distribution, whereas the latter focuses on data-flow coordination.

In [19], the synthesis of a distributed controller is made decidable by model-checking *knowledge* properties and exploiting *temporary synchronization*. Similar concepts have been exploited also for monitoring global properties in distributed systems [18]. Model-checking knowledge properties for addressing scheduling problems in distributed systems has been exploited also in [6]. The work described in [34] shares the same idea of using additional interactions in order to synthesize "joint" knowledge of several concurrent and distributed entities. Our notions of CM and coordination information resemble their notions of knowledge and temporary synchronization, respectively.

Priorities define precedence relations between component actions. Since priorities can be used to restrict the behaviour of a system in order to avoid undesired states, our work is related to priority scheduling. In [12], a priority synthesis algorithm is presented. It aims at synthesizing a system that is deadlock-free or satisfies some safety property. In [13], the authors describe an algorithm to synthesize local sets of priorities hence addressing distribution of the control logic. Priority-based synthesis of distributed controllers has been addressed also in [9] and, in the Web service orchestration setting, in [36]. These works have been either conceived in the context of the *BIP framework*, which is quite different from ours, or adopting its main concepts and operations.

8 Conclusions and future work

We have formalized a method for the distributed enforcement of service choreographies. It takes as input a choreography specification given in the form of an automata-based model, called CeFM, derived from a BPMN2 choreography diagram. This specification is automatically decomposed into a set of CMs that contain the information, local to each participant service, needed to coordinate the global interaction of all the participants from outside so to realize the specified choreography. Relying on the generated CMs, a distributed coordination algorithm has been defined with the aim of characterizing the coordination logic that has to be performed by additional software entities, called CDs. Once interposed among the participant services, CDs collaborate with each other to enforce the specified choreography.

We have illustrated the applicability of our method by means of a distributed social proximity network scenario. We have proven correctness of the defined algorithm with respect to preventing those interactions that do not belong to the choreography specification. To this end, we have rigorously characterized the notion of undesired interaction.

The formalized method is implemented as part of a model-based tool chain released under the FISSi initiative (`http://www.ow2.org/view/Future_Internet`) to support the development of choreography-based systems in the CHOReOS EU project. The proposed method has been also applied to other real-world use cases in the *Airport*, and *Marketing and Sale* domains. Refer to the

CHOReO*Synt* web site `http://choreos.disim.univaq.it` for documentation. The application to these use cases has shown that the method is viable and the overhead due to the exchange of synchronization information is negligible. Indeed, in the worst case, it is bound by the number of tasks times the number of participants.

Beyond the modelling of message-based interaction, BPMN2 Choreography Diagrams provide constructs for specifying events. As future work, we plan to extend our definitions of CeFM, and related CMs, to account for event-based coordination. As a consequence, the distributed coordination algorithm formalized in Section 5 must also be revised. As mentioned in Section 1, our distributed coordination algorithm is general in the sense that it is service-independent. As a further future work, we will leverage our preliminary results in [4, 23] to release the assumption that the participants already match the corresponding role specifications. This means that we have to define an extended version of CDs that are able to address both coordination and adaptation issues in a modular manner, hence still promoting separation of concerns. Moreover, we plan to prove other notions of correctness beyond undesired interaction prevention, e.g., progress or permissiveness, and deadlock freedom beyond deadlocking joins. Last but not least, not all choreographies can be enforced by our method and, as further future work, we should formally characterize the conditions under which a choreography is enforceable by our method.

Acknowledgments

This work is supported by the Ministry of Education, Universities and Research, prot. 2012E47TM2 (project IDEAS - Integrated Design and Evolution of Adaptive Systems), and was supported by the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement number 257178 (project CHOReOS - Large Scale Choreographies for the Future Internet - `www.choreos.eu`).

We acknowledge the anonymous reviewers for their thoughtful feedbacks and valuable suggestions that allowed us to significantly improve the paper.

References

[1] Farhad Arbab, 6 Christel Baier, Frank S. de Boer, Jan J. M. M. Rutten & Marjan Sirjani (2005): *Synthesis of Reo Circuits for Implementation of Component-Connector Automata Specifications*. In: *COORDINATION*, pp. 236–251, doi:10.1007/11417019_16.

[2] Farhad Arbab, Natallia Kokash & Sun Meng (2008): *Towards Using Reo for Compliance-Aware Business Process Modeling*. In: *ISoLA*, pp. 108–123, doi:10.1007/978-3-540-88479-8_9.

[3] Marco Autili, Davide Ruscio, Amleto Salle, Paola Inverardi & Massimo Tivoli (2013): *A Model-Based Synthesis Process for Choreography Realizability Enforcement*. In: *FASE*, *LNCS* 7793, Springer Berlin Heidelberg, pp. 37–52, doi:10.1007/978-3-642-37057-1_4.

[4] Marco Autili, Amleto Salle & Massimo Tivoli (2013): *Synthesis of Resilient Choreographies*. In: *Software Engineering for Resilient Systems*, LNCS 8166, Springer Berlin Heidelberg, pp. 94–108, doi:10.1007/978-3-642-40894-6_8.

[5] Christel Baier, Joachim Klein & Sascha Klppelholz (2011): *A Compositional Framework for Controller Synthesis*. In: *CONCUR*, *LNCS* 6901, Springer Berlin Heidelberg, pp. 512–527, doi:10.1007/978-3-642-23217-6_34.

[6] Ananda Basu, Saddek Bensalem, Doron Peled & Joseph Sifakis (2009): *Priority Scheduling of Distributed Systems Based on Model Checking*. In: *Computer Aided Verification*, *LNCS* 5643, Springer Berlin Heidelberg, pp. 79–93, doi:10.1007/978-3-642-02658-4_10.

[7] Samik Basu & Tevfik Bultan (2011): *Choreography conformance via synchronizability*. In: *WWW*, pp. 795–804, doi:10.1145/1963405.1963516.

[8] Samik Basu, Tevfik Bultan & Meriem Ouederni (2012): *Deciding choreography realizability*. In: *POPL*, ACM, pp. 191–202, doi:10.1145/2103656.2103680.

[9] Imene Ben-Hafaiedh, Susanne Graf & Sophie Quinton (2011): *Building Distributed Controllers for Systems with Priorities*. *The Journal of Logic and Algebraic Programming* 80(3 - 5), pp. 194 – 218, doi:10.1016/j.jlap.2010.10.001.

[10] Antonio Brogi & Razvan Popescu (2006): *Automated Generation of BPEL Adapters*. In: *In Proc. of ICSOC'06, volume 4294 of LNCS*, Springer, pp. 27–39.

[11] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella & Fabio Patrizi (2008): *Automatic Service Composition and Synthesis: the Roman Model*. *IEEE Data Eng. Bull.* 31(3), pp. 18–22.

[12] Chih-Hong Cheng, Saddek Bensalem, Yu-Fang Chen, Rongjie Yan, Barbara Jobstmann, Harald Ruess, Christian Buckl & Alois Knoll (2011): *Algorithms for Synthesizing Priorities in Component-Based Systems*. In: *Automated Technology for Verification and Analysis*, *LNCS* 6996, Springer Berlin Heidelberg, pp. 150–167, doi:10.1007/978-3-642-24372-1_12.

[13] Chih-Hong Cheng, Saddek Bensalem, Rongjie Yan & Harald Ruess (2011): *Distributed Priority Synthesis and its Applications*. *CoRR* abs/1112.1783.

[14] CHOReOS Project Team (2013): *Final CHOReOS Architectural Style and its Relation with the CHOReOS Development Process and IDRE - Public Project deliverable D1.4 (b)*.

[15] Christel Baier and Joachim Klein and Sascha Klppelholz (2011): *Synthesis of Reo Connectors for Strategies and Controllers*. In: *LAM*, doi:10.1016/j.scico.2010.03.002.

[16] Jörg Desel & Javier Esparza (1995): *Free Choice Petri Nets*. Cambridge University Press, New York, NY, USA, doi:10.1017/CBO9780511526558.

[17] Gregor Gössler & Gwen Salaün (2012): *Realizability of Choreographies for Services Interacting Asynchronously*. In: *FACS*, *LNCS* 7253, pp. 151–167, doi:10.1007/978-3-642-35743-5_10.

[18] Susanne Graf, Doron Peled & Sophie Quinton (2011): *Monitoring Distributed Systems Using Knowledge*. In: *Formal Techniques for Distributed Systems*, *LNCS* 6722, Springer Berlin Heidelberg, pp. 183–197, doi:10.1007/978-3-642-21461-5_12.

[19] Susanne Graf, Doron Peled & Sophie Quinton (2012): *Achieving distributed control through model checking*. *Formal Methods in System Design* 40(2), pp. 263–281, doi:10.1007/s10703-011-0138-9.

[20] Matthias Güdemann, Pascal Poizat, Gwen Salaün & Alexandre Dumont (2013): *VerChor: A Framework for Verifying Choreographies*. In: *FASE*, *LNCS* 7793, pp. 226–230, doi:10.1007/978-3-642-37057-1_16.

[21] Matthias Güdemann, Gwen Salaün & Meriem Ouederni (2012): *Counterexample guided synthesis of monitors for realizability enforcement*. In: *ATVA*, LNCS, pp. 238–253, doi:10.1007/978-3-642-33386-6_20.

[22] Sylvain Hallé & Tevfik Bultan (2010): *Realizability analysis for message-based interactions using shared-state projections*. In: *FSE*, pp. 27–36, doi:10.1145/1882291.1882298.

[23] Paola Inverardi & Massimo Tivoli (2013): *Automatic synthesis of modular connectors via composition of protocol mediation patterns*. In: *Proc. of ICSE'13*, pp. 3–12.

[24] Sung-Shik T. Q. Jongmans & Farhad Arbab (2013): *Global Consensus through Local Synchronization*. In: *ESOCC Workshops*, pp. 174–188, doi:10.1007/978-3-642-45364-9_15.

[25] Sung-Shik T. Q. Jongmans, Francesco Santini & Farhad Arbab (2014): *Partially-Distributed Coordination with Reo*. In: *PDP*, pp. 697–706, doi:10.1109/PDP.2014.19.

[26] Sung-Shik T. Q. Jongmans, Francesco Santini, Mahdi Sargolzaei, Farhad Arbab & Hamideh Afsarmanesh (2012): *Automatic Code Generation for the Orchestration of Web Services with Reo*. In: *ESOCC*, pp. 1–16, doi:10.1007/978-3-642-33427-6_1.

[27] Natallia Kokash & Farhad Arbab (2008): *Formal Behavioral Modeling and Compliance Analysis for Service-Oriented Systems*. In: *FMCO*, pp. 21–41, doi:10.1007/978-3-642-04167-9_2.

[28] Natallia Kokash & Farhad Arbab (2013): *Formal Design and Verification of Long-Running Transactions with Extensible Coordination Tools*. *IEEE T. Services Computing* 6(2), pp. 186–200, doi:10.1109/TSC.2011.46.

[29] Leslie Lamport (1978): *Time clocks, and the ordering of events in a distributed system*. *Commun. ACM* 21, pp. 558–565, doi:10.1145/359545.359563.

[30] Annapaola Marconi, Marco Pistore & Paolo Traverso (2008): *Automated Composition of Web Services: the ASTRO Approach*. *IEEE Data Eng. Bull.* 31(3), pp. 23–26.

[31] Tarek Melliti, Pascal Poizat & Sonia Ben Mokhtar (2008): *Distributed behavioural adaptation for the automatic composition of semantic services*. In: *FASE*, pp. 146–162, doi:10.1007/978-3-540-78743-3_12.

[32] Sun Meng, Farhad Arbab & Christel Baier (2011): *Synthesis of Reo circuits from scenario-based interaction specifications*. *Sci. Comput. Program.* 76(8), pp. 651–680, doi:10.1007/978-3-540-78743-3_12.

[33] Jyotishman Pathak, Robyn Lutz & Vasant Honavar (2008): *MoSCoE: An Approach for Composing Web Services through Iterative Reformulation of Functional Specifications*. *International Journal on Artificial Intelligence Tools* 17, pp. 109–138, doi:10.1142/S0218213008003807.

[34] Doron Peled & Sven Schewe (2011): *Practical Distributed Control Synthesis*. In: *INFINITY*, *EPTCS* 73, pp. 2–17, doi:10.4204/EPTCS.73.2.

[35] Pascal Poizat & Gwen Salaün (2012): *Checking the Realizability of BPMN 2.0 Choreographies*. In: *SAC*, pp. 1927–1934, doi:10.1145/2245276.2232095.

[36] S. Quinton, I. Ben-Hafaiedh & S. Graf (2009): *From Orchestration to Choreography: Memoryless and Distributed Orchestrators*. In: *FLACOS*.

[37] Gwen Salaün (2008): *Generation of Service Wrapper Protocols from Choreography Specifications*. In: *SEFM*, pp. 313–322, doi:10.1109/SEFM.2008.42.

[38] Koushik Sen, Abhay Vardhan, Gul Agha & Grigore Rosu (2004): *Efficient Decentralized Monitoring of Safety in Distributed Systems*. In: *Proc. of ICSE*.

[39] Jocelyn Simmonds, Yuan Gan, Marsha Chechik, Shiva Nejati, Bill O'Farrell, Elena Litani & Julie Waterhouse (2009): *Runtime Monitoring of Web Service Conversations*. *IEEE T. Services Computing* 2(3), doi:10.1109/TSC.2009.16.

[40] Jianwen Su, Tevfik Bultan, Xiang Fu & Xiangpeng Zhao (2007): *Towards a Theory of Web Service Choreographies*. In: *WS-FM*, pp. 1–16, doi:10.1007/978-3-540-79230-7_1.